

课程设计（论文）

课程所属学院 信息与软件工程学院

所属课程名称 数据库原理及应用课程设计

项目　　名称 学院生活平台

院　　（系） 计算机科学与技术（数据工程）系

班　　级 [redacted]

学　　生 0103 [redacted]

指 导 教 师 喻佳

辅 导 教 师 喻佳

2026　年　5　月　2　日

课程设计(论文)任务书

天佑 学　院　计算机 2023　专　业　1　班

一、课程设计(论文)题目：　学院生活平台

二、课程设计(论文)工作：

自 2026 年 4 月 15 日起至 2026 年 5 月 2 日止

三、课程设计(论文) 地点：　26-611、　图书馆

四、课程设计(论文)内容要求：

1. 本课程设计的目的

(1) 使学生掌握数据库开发的基本工作流程：能够运用基本科学原理，借助文献研究，确定本课程设计的选题，并给出对应选题的多种设计方案，通过讨论、分析确定设计的主要内容及需求分析。

(2) 巩固数据库课程所学的内容：熟练使用 SQL 语句，掌握构建数据库管理系统的六大过程并能够构建一个完整的数据库系统（包括数据库和数据库管理系统；

(3) 培养学生的计算机思维能力以及合作的精神：养成良好职业素养 (如专业能力、管理决策能力以及团队协作能力等)，具有自主学习意识和能力；

(4) 培养学生分析、解决问题的能力：整理、分析实验结果，运用数据库原理的相关理论完成设计可行性报告及课程设计论文（含需求分析、概要设计、详细设计等开发过程）等的撰写；

(5) 提高学生的科技论文写作能力。

2. 课程设计的任务及要求

1) 基本要求：

(1) 研究课程设计任务，并进行系统需求分析；

(2) 对系统进行总体设计，分解系统功能模块，进行任务分配，以实现分工合作；

(3) 实现各功能模块代码；

(4) 组装各模块，并测试、完善系统。

2) 创新要求：

在基本要求达到后，可进行创新设计，如改进界面、增加功能或进行代码优化。

3) 课程设计论文编写要求

(1) 要按照书稿的规格打印誊写课程设计论文

(2) 论文包括封面、设计任务书（含评语）、摘要、目录、设计内容、设计小结

(3) 论文装订按学校的统一要求完成

4) 参考文献：

(1) 王珊.数据库系统概论（第六版）.北京.高等教育出版社,2024

(2) 孟小峰, 周龙骧, 王珊. 数据库技术发展趋势[J]. 软件学报, 2004

(3) https://www.postgresql.org/docs/16/ddl-constraints.html

(4) https://www.postgresql.org/docs/16/indexes-multicolumn.html

(5) https://www.postgresql.org/docs/16/indexes-partial.html

(6) https://www.postgresql.org/docs/16/sql-createtrigger.html

(7) https://www.postgresql.org/docs/16/rangetypes.html

(8) https://www.postgresql.org/docs/16/btree-gist.html

(9) Elmasri R, Navathe S B. Fundamentals of database systems. 7th ed. Pearson, 2016.

(10) Silberschatz A, Korth H F, Sudarshan S. Database system concepts, 2019

5) 课程设计进度安排

内容	天数	地点
系统总体设计	2	宿舍
软件设计及调试	4	实验室、图书馆
撰写报告	2	实验室、图书馆
答辩	2	实验室

学生签名： _____

2026 年 5 月 2 日

课程设计(论文)评审意见

(1) 课程设计过程 (20 分)：优（ ）、良（ ）、中（ ）、一般（ ）、差（ ）；
(2) 是否完成调试，系统运行效果 (20 分)：优（ ）、良（ ）、中（ ）、一般（ ）、差（ ）；
(3) 回答问题 (30 分)：优（ ）、良（ ）、中（ ）、一般（ ）、差（ ）；
(4) 课程设计报告 (30 分)：优（ ）、良（ ）、中（ ）、一般（ ）、差（ ）；
(5) 格式规范性及考勤是否降等级：是（ ）、否（ ）

评阅人：喻佳 **职称**：副教授

2026 年 5 月 10 日

学院生活平台

摘要：

本课程设计以已有的学院生活平台为基础，选取平台基础、功能房预约和课程资源分享三个子系统，重新梳理其中的数据库设计。报告依次说明需求分析、概念结构设计、逻辑结构设计、物理结构设计、数据库对象实现和运行验证，主要讨论实体联系抽象、关系模式转换、函数依赖与规范化、完整性约束、典型 SQL，以及事务和并发控制的处理方式。

概念结构设计部分整理了用户身份、用户资料、部门层级、角色权限、数据范围、房间资源、预约事务、课程资源、下载事件和积分事件等对象，并绘制总体分层图、子系统 E-R 图和主要业务流程图。逻辑结构设计部分围绕主要关系模式说明主键、候选键、主属性、非主属性、函数依赖和范式判断，并单独讨论部门闭包表、审计快照字段以及课程资源中的受控冗余。

物理设计与实现部分使用 PostgreSQL 落地显式物理外键、部分唯一索引、复合外键、检查约束、排斥约束、延迟约束触发器、审计表阻断触发器和资料同步函数等对象。功能房预约模块用 EXCLUDE USING gist 约束同一房间的活跃预约时间区间；课程资源模块用复合外键检查资源所属专业、积分归属作者和最佳推荐状态；平台基础模块则用闭包表、模块字典、能力表和审计快照保存组织权限关系及历史记录。

运行验证部分结合 PostgreSQL 实库迁移记录，检查了预约时间冲突、参与人数下限、申请人与参与人一致性、资源专业归属一致性、资源状态一致性、最佳推荐约束、积分事件作者一致性、模块能力外键以及审计 JSON 结构等规则。测试结果表明，数据库可以直接拒绝这些非法写入，本设计中的主要约束已经能在实际运行中生效。

目 录

摘要 .....

第一章 设计目的 1

1.1 设计背景 1

1.2 设计目标 1

1.3 设计范围 1

1.4 数据库设计原则 2

第二章 设计需求分析 3

2.1 业务角色与使用场景 3

2.2 总体需求分层 3

2.3 平台基础子系统的数据需求 4

2.3.1 身份与资料分离需求 4

2.3.2 组织结构与数据范围需求 4

2.3.3 RBAC 与审计需求 5

2.4 功能房预约子系统的数据需求 5

2.5 课程资源分享子系统的数据需求 5

2.6 核心业务规则分析 6

2.7 需求到数据库设计的映射 7

第三章 概要设计 8

3.1 总体概念结构设计 8

3.2 平台基础子系统 E-R 设计 8

3.2.1 身份、资料与组织结构 8

3.2.2 RBAC 与数据范围配置 9

3.2.3 审计与配置支撑结构 10

3.3 功能房预约子系统 E-R 设计 10
3.4 课程资源分享子系统 E-R 设计 10
3.5 E-R 模型到关系模式的转换 11
3.6 主要关系模式设计 11
3.7 规范化分析 12
3.7.1 核心关系的函数依赖分析 12
3.7.2 受控冗余与规范化取舍 15
3.8 物理对象规划概览 16
第四章 详细设计 17
4.1 平台基础子系统详细设计 17
4.1.1 新用户创建与资料同步流程 17
4.1.2 部门树维护与闭包表更新 18
4.1.3 角色数据范围整体替换事务 19
4.2 功能房预约子系统详细设计 19
4.2.1 预约创建与并发控制流程 19
4.2.2 预约核心 DDL 与 DML 设计 21
4.2.3 典型复杂查询 21
4.3 课程资源分享子系统详细设计 22
4.3.1 资源投稿、审核与发布流程 22
4.3.2 资源核心约束与受控冗余 23
4.3.3 最佳推荐与状态触发器 24
4.3.4 典型聚合查询 24
4.4 SQL 设计汇总 24
第五章 设计结果/运维 25
5.1 平台基础子系统运行结果 25
5.1.1 新用户创建与资料同步结果 25

5.1.2 部门树维护结果 25
5.1.3 角色配置结果 26
5.2 功能房预约子系统运行结果 27
5.2.1 合法预约提交结果 27
5.2.2 冲突预约拦截结果 27
5.2.3 预约审核后的状态变更结果 28
5.3 课程资源分享子系统运行结果 29
5.3.1 资源投稿与审核发布结果 29
5.3.2 课程资源积分榜查询结果 29
5.4 本章小结 30
第六章 总结 31
6.1 主要收获 31
6.2 当前不足 31
6.3 后续改进方向 32
参考文献 33

第一章　设计目的

1.1　设计背景

数据库课程设计要求从实际业务出发，完成数据建模、关系模式设计、数据库对象实现和验证分析。学院生活平台已经经过多轮迭代，业务内容较多。本报告结合当前可运行的功能范围，选取平台基础、功能房预约和课程资源分享三个子系统展开说明。其中，用户身份、用户资料、角色权限、数据范围、组织结构和审计日志归入平台基础子系统，为另外两个业务子系统提供共用的数据基础。

1.2　设计目标

本课程设计主要完成四项工作：从业务需求中抽取稳定的数据对象和联系，建立与实际业务过程相符的关系结构；围绕主要关系模式给出主键、候选键、外键、函数依赖、范式判断和删除策略；将适合由数据库承担的业务规则写成约束、索引、触发器或事务逻辑；最后结合迁移记录和测试结果说明设计的运行效果。

围绕这些目标，报告重点分析部门闭包表、角色与数据范围复合外键、房间预约时间冲突排斥约束、资源审核状态一致性约束、最佳推荐触发器以及审计日志 append-only 机制。这些对象也是本次概念设计、逻辑设计和物理实现中最容易出错的几处。

1.3　设计范围

本报告的设计范围如下。

表 1　报告设计范围

设计内容	主要内容	说明
平台基础子系统	用户资料、部门树、岗位、角色、权限、数据范围、审计日志、配置	构成三个子系统共用的数据基础与控制结构
功能房预约子系统	楼房、房间、预约、参与人、封禁、时间冲突控制	用于说明事务管理、状态一致性与并发控制
课程资源分享子系统	专业、课程、资源、最佳推荐、下载事件、积分事件	用于说明教学主数据、审核状态与事件事实建模

后续 E-R 图、关系模式、SQL 设计和测试验证都围绕这三个子系统展开。主键、外键、范式、约束和事务分析也都对应表中的具体业务内容。

1.4 数据库设计原则

数据库设计既要满足用户的信息需求和处理需求，也要让数据结构准确表达业务对象及其联系。冗余、数据独立性和后续维护成本也需要一起考虑。按照一般设计过程，本报告从需求分析开始，依次完成概念结构设计、逻辑结构设计和物理结构设计。本次设计主要遵循以下原则。

第一，表结构先跟业务对象对齐，而不是由界面形式或程序写法反推。只有先明确用户资料、组织结构、房间预约、课程资源等业务内容之间的真实联系，后续 E-R 模型、主外键设计和约束设置才有依据。第二，关系模式尽量满足规范化要求，减少不必要的数据重复，降低插入、删除和更新异常出现的可能性。本课程设计中的主体表和桥接表大多按照第三范式或更高范式组织；对于部门闭包表、审计快照字段以及课程资源中的受控冗余，则保留其业务必要性，并用外键、触发器和一致性检查维持数据正确性。

第三，数据库不只是保存数据，还要把数据限制在合法状态内。设计中需要综合运用主键、外键、唯一约束、检查约束、触发器和事务机制，对实体完整性、参照完整性和用户定义完整性进行控制。对于多个业务过程共同依赖的数据，应采用统一而稳定的组织方式，提高数据共享程度，并减少结构分散带来的维护成本。平台基础子系统中的用户身份、用户资料、部门、角色、权限和数据范围等内容，正是按这一原则组织，进而为功能房预约和课程资源分享两个业务子系统提供一致的数据支撑。

最后，数据库设计还要考虑运行效率和维护成本。在保证正确性的前提下，应针对高频查询、层级检索和时间冲突检测设置索引、排斥约束等物理对象；后续结构调整则通过迁移脚本、配置表和清晰的关系边界来控制，避免数据库越改越乱。

第二章　设计需求分析

2.1　业务角色与使用场景

本课程设计以学院生活平台为对象，关注业务活动中涉及的数据对象、约束关系和状态变化。结合 roles 表中的预置角色以及本次实现范围，主要业务角色及职责见表 2。

表 2　业务角色与主要职责

业务角色	角色编码	主要职责
普通用户	user	维护个人资料，提交功能房预约，上传课程资源，并完成资源的下载与查看。新用户注册后默认写入该角色。
工作人员	staff	在本次课程设计范围内主要承担功能房预约的日常审核工作，包括查看预约申请，并执行通过或驳回等处理。
管理员	admin	负责平台基础数据维护与跨模块业务管理，包括部门、岗位、角色、权限、平台配置和审计日志等内容，并对功能房预约和课程资源进行全局管理。
超级管理员	super_admin	系统最高权限角色，负责平台级全局权限控制与重要管理操作，对管理员职责形成统一兜底。
专业负责人	major_lead	在所负责专业范围内维护课程信息，审核课程资源，执行下架、统计和最佳推荐等操作，其业务范围由专业负责人映射关系限定。

这样划分角色，是为了明确不同用户能看哪些数据、能做哪些操作，以及操作后如何追溯。普通用户、工作人员、管理员和超级管理员主要通过 user_roles 与 role_permissions 授权；专业负责人除了角色身份，还要通过 major_leads 表绑定到具体专业。预约审核、资源审核、组织维护和全局配置的权限边界不同，后续关系模式、约束、索引和事务设计都要围绕这些边界处理。

2.2　总体需求分层

需求分析按三条线展开：平台基础层、预约事务层和资源事务层。图 1 给出了总体分层。

图 1 中，功能房预约和课程资源分享都依赖平台基础子系统。预约申请人、资源作者、角色权限和数据范围过滤，都要先有身份、组织和权限结构作为前提；底层的数据库设计则围绕完整性约束、较高级别范式、SQL 设计、索引设计以及事务并发控制展开，所以本章先说明这些共用需求。

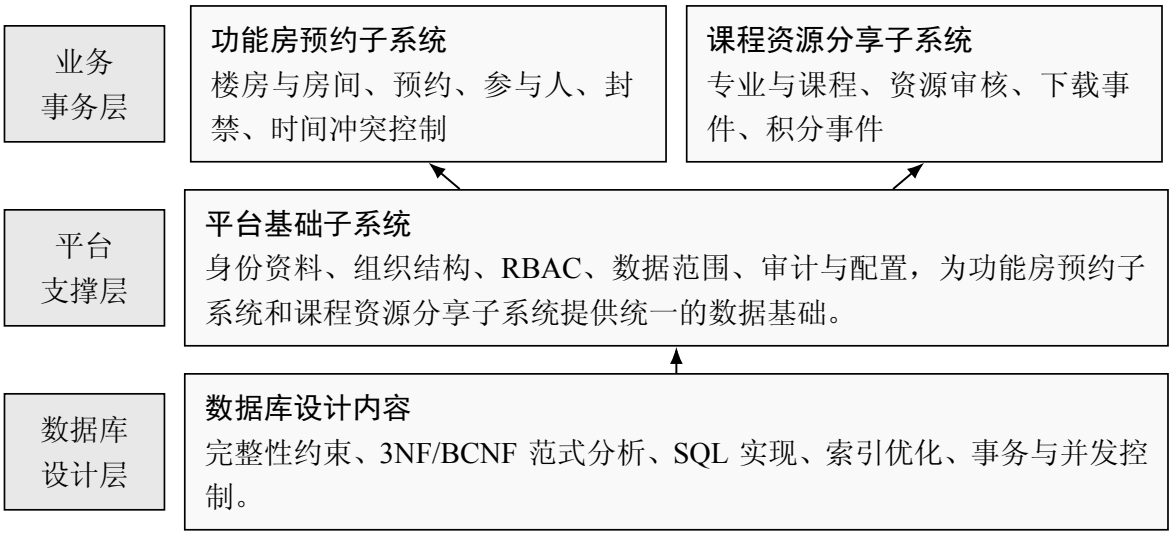


图 1　三条主线模块的总体需求分层

2.3　平台基础子系统的数据需求

平台基础子系统承担了身份、组织、权限和审计四类需求。

2.3.1　身份与资料分离需求

当前项目以 auth.users 作为身份事实源，以 profiles 作为业务资料表。这样拆分主要有三个原因：

1. 用户认证信息和业务资料扩展字段的生命周期不同；
2. 用户资料需要保存姓名、学号、状态、登录时间等业务属性；
3. 后续又新增了邮箱、邮箱确认时间、封禁快照、删除快照等同步字段，说明资料表必须支持演化。

所以，用户建模拆成“身份事实源”和“业务资料扩展”两类对象，再用一对一关系衔接。

2.3.2　组织结构与数据范围需求

在当前实现中，用户与部门之间用 user_departments 表达多对多关系。系统还要支持“本部门”“本部门及子部门”“自定义部门集合”等数据范围。所以，部门结构既要表达直接父子关系，又要支持对整棵子树的范围展开。

对应到数据库，主要有以下三项要求：

1. departments 自引用表示直接上下级；
2. department_closure 表示祖先到后代的可达关系；
3. role_data_scopes 与 role_data_scope_departments 表示角色在模块维度上的数据范围。

2.3.3 RBAC 与审计需求

平台基础层除标准 RBAC 结构外，还要求对管理行为进行追溯。系统除了 roles、permissions、user_roles 与 role_permissions，还需要建立 audit_logs 表，用于保存操作者标识、操作对象、操作结果以及事件发生时的身份快照，避免后续资料变更破坏历史语义。

2.4 功能房预约子系统的数据需求

功能房预约模块属于典型的时间资源管理场景，需要记录几类数据。

1. 楼房与房间是稳定的主数据，房间必须依附楼房存在。
2. 预约记录要保存房间、申请人、开始时间、结束时间、审核信息、取消信息和用途说明。
3. 一条预约不是单人行为，必须有参与人集合，其中恰有一名申请人。
4. 相同房间在有效预约状态下不能出现时间区间重叠。
5. 对违规用户建立独立封禁记录，用于保存处理过程与封禁历史。

这些需求说明，预约模块不能只靠一张“预约总表”概括全部业务事实，而应分别建立楼房、房间、预约主体、参与人及封禁记录等关系，并用约束和事务维持一致性。

2.5 课程资源分享子系统的数据需求

课程资源分享模块同时涉及教学层级、审核状态、推荐标签与事件事实，层次比预约模块更明显。

1. 资源必须依附课程，而课程又必须依附专业；
2. 一个专业可以配置多名负责人，一个负责人也可能负责多个专业；
3. 资源存在草稿、待审、已发布、已驳回、已下架等状态流转；
4. 文件资源和外链资源的字段组合不同，且同一课程下需要去重；
5. 下载统计和积分统计通过事实表支撑排行榜计算与审计追溯。

所以，本模块除 majors、courses、course_resources 等主体表外，还需要围绕资源事件设计若干事实表，包括 course_resource_bests、course_resource_download_events 与 course_resource_score_events，用来区分资源主体与围绕资源发生的事件。

2.6 核心业务规则分析

结合需求文档、数据库迁移脚本与当前实现，可以把三条主线中的主要业务规则整理为表 3。这些规则分别对应结构完整性、状态一致性、时间冲突控制和统计语义保持等数据库问题。先把规则列清楚，后文的主外键、检查约束、索引和触发器才有来源。

表 3　核心业务规则汇总

所属模块	业务规则	对数据库设计的直接影响
平台基础	一个用户可属于多个部门	采用 user_departments 桥接表表达多部门归属
平台基础	部门范围要支持子树展开	采用 department_closure 闭包表
平台基础	审计日志必须保留历史真实性	审计表 append-only，保留操作者快照字段
功能房预约	同一房间有效预约不能重叠	采用时间区间排斥约束与部分索引
功能房预约	每条预约至少 3 名参与人	采用延迟约束触发器校验
功能房预约	申请人必须在参与人集合中唯一出现	采用部分唯一索引与触发器协同约束
课程资源	文件与外链资源字段必须互斥	采用类型相关的 CHECK 约束
课程资源	资源状态与审核字段组合必须一致	采用状态一致性 CHECK 约束
课程资源	资源最佳推荐仅限已发布资源	采用发布态检查触发器与状态变更清理触发器
课程资源	积分事件必须绑定资源作者和资源所属专业	采用复合外键锁定受控冗余一致性

表 3 中的规则，会直接落到后续索引、约束、触发器和事务设计中。本设计不是先选技术手段再寻找场景，而是先明确业务边界，再把边界写成可执行的数据库约束。

2.7 需求到数据库设计的映射

表 4 整理了代表性需求和数据库实现思路。这里关注的不只是“哪张表对应哪项需求”，还包括逻辑层和物理层如何衔接。例如，“范围控制”或“唯一性保证”在逻辑层可能表现为联系拆分，在物理层则落到复合键、部分索引或触发器上。

表 4 需求到数据库设计的映射

需求表述	关系模式层的设计	物理层的实现
查询本部门及全部子部门用户	departments + department_closure + user_departments	祖先索引、后代索引、闭包维护触发器
角色在模块维度上配置数据范围	app_modules + data_scope_modules + role_data_scopes	模块字典外键、复合主键、复合外键
同房间不能出现时间冲突	facility_reservations	EXCLUDEUSINGgist、部分索引
预约主体与参与人原子提交	facility_reservations + facility_reservation_participants	事务写入、延迟约束触发器
资源按课程维度去重	course_resources	文件哈希唯一索引、规范化 URL 唯一索引
资源按专业统计与作者积分排名	course_resources + course_resource_score_events	复合外键、聚合查询索引、事件事实表

表 4 的对应关系说明，需求分析中提出的问题，后续都可以落到具体的关系模式和数据库对象上。这样处理后，“需求分析”“概要设计”和“详细设计”三部分之间的衔接也更清楚。

第三章　概要设计

3.1　总体概念结构设计

沿着前一章的需求分析，数据库对象可以分为三类：平台基础主数据、业务事务主表和事件事实表。平台基础层保存身份、资料、组织、权限、数据范围和审计追踪等相对稳定的信息；业务事务层保存功能房预约、课程资源等会经历申请、审核、发布、取消等状态变化的主体；事件事实层记录参与、下载、积分和最佳推荐等行为。这样划分后，不同性质的数据不会挤在同一张表里，也能减少空值字段和关系边界不清的问题。

3.2　平台基础子系统 E-R 设计

3.2.1　身份、资料与组织结构

平台基础子系统首先需要回答“人从哪里来、资料放在哪里、组织关系如何表达、部门范围如何展开”这四个问题。图 2 给出了身份、资料和组织结构的 E-R 设计。

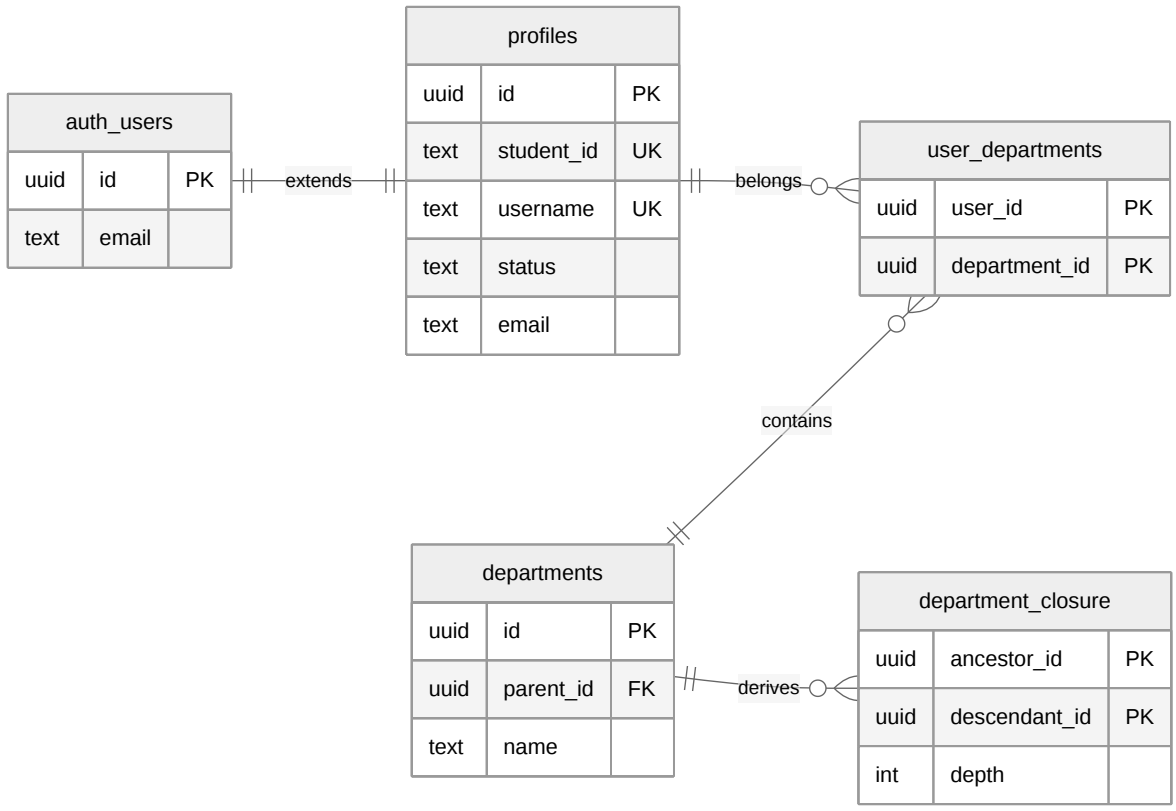


图 2　平台基础子系统 E-R 图（一）：身份、资料与组织结构

图 2 中，profiles 与 auth.users 采用共享主键的一对一扩展关系，用来拆开认证身份和业务资料；用户归属部门用 user_departments 建模为多对多关系，以适配多部门归属场景；department_closure 则是面向“部门及子部门”查询的派生关系表，与 departments

的自引用结构配合使用。

3.2.2 RBAC 与数据范围配置

平台基础层的第二个重点是 RBAC 与数据范围设计。如果只建用户、角色、权限三类实体，模型仍然停留在普通 RBAC；本项目还要求在模块维度上配置数据范围，所以又引入了模块字典、能力表和自定义部门映射，如图 3 所示。

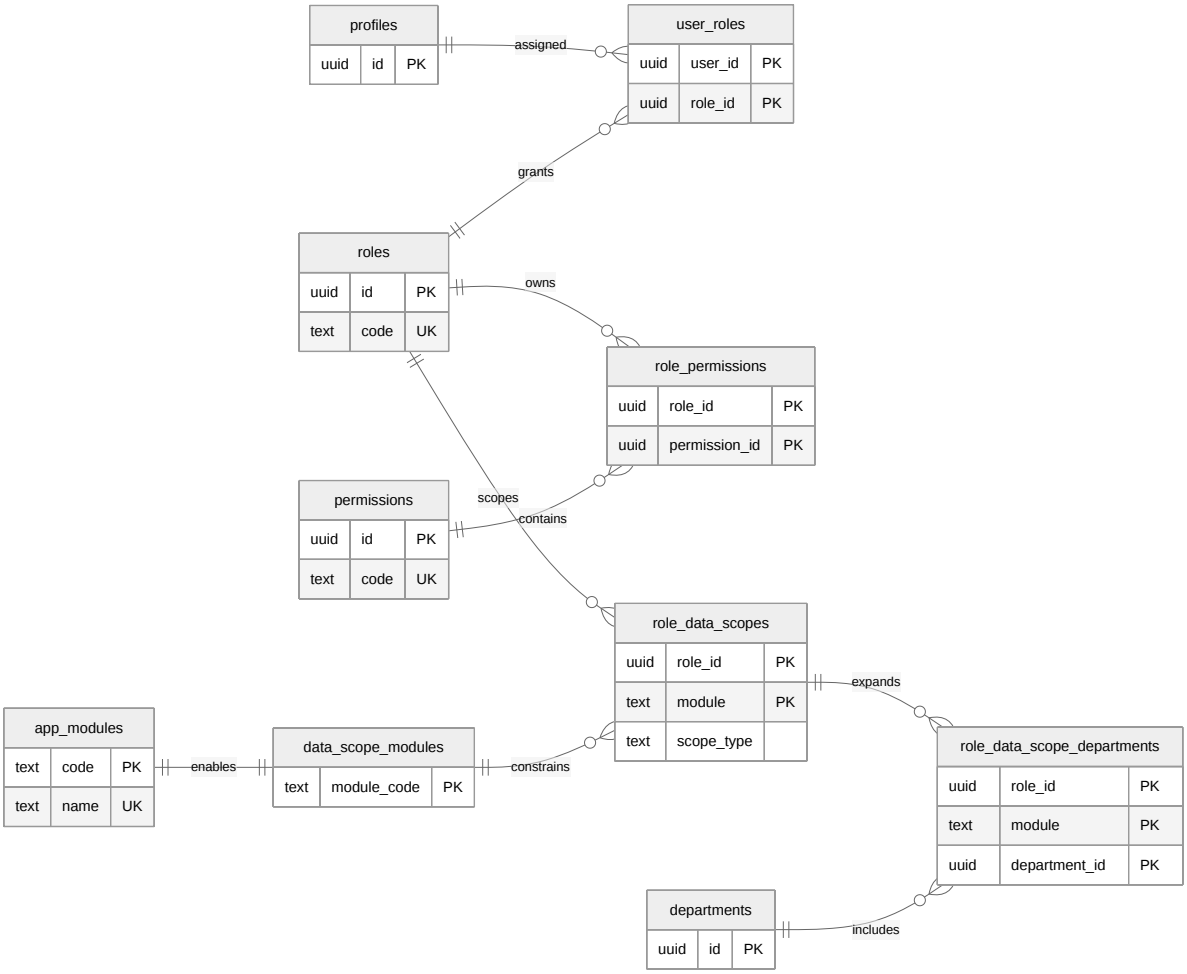


图 3　平台基础子系统 E-R 图（二）：RBAC 与数据范围

图 3 中需要注意两处。user_roles 与 role_permissions 都是多对多拆分表，关系边界清楚。app_modules 与 data_scope_modules 分别表达系统模块和数据范围能力；role_data_scopes.module 由能力表外键约束。

3.2.3 审计与配置支撑结构

除上述主体外，平台基础层还包含 audit_logs 和 app_config 两张支撑表。前者是 append-only 审计表，后者是平台配置表。审计日志同时保存 actor_user_id 与 actor_name、actor_email、actor_roles 等快照字段，用于保留事件发生时的历史语义。

3.3 功能房预约子系统 E-R 设计

功能房预约模块围绕空间资源、预约事务和参与人一致性展开，其 E-R 图如图 4 所示。

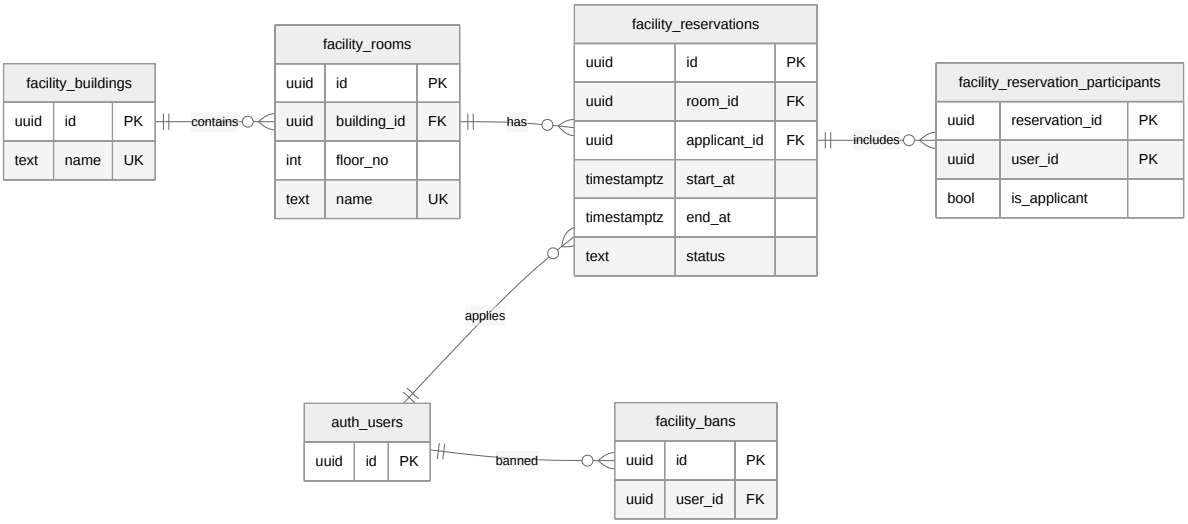


图 4　功能房预约子系统 E-R 图

图 4 所示模型的约束比较集中。预约主体 facility_reservations 同时包含时间区间、状态、审核信息和取消信息；参与人表 facility_reservation_participants 还要与主表中的 applicant_id 保持一致。所以，该模块主要处理跨行一致性和并发冲突控制。

3.4 课程资源分享子系统 E-R 设计

课程资源模块的 E-R 设计如图 5 所示。与预约模块相比，它更强调教学主数据和事件事实的分层。

图 5 中容易引起疑问的是受控冗余：course_resources 中的 major_id，以及 course_resource_score_events 中的 major_id、user_id，都可以由课程或资源主体间接推出。该模块没有让这些字段自由重复，而是用复合唯一键与复合外键做一致性约束，所以仍然保留了规范化分析所要求的依赖控制。

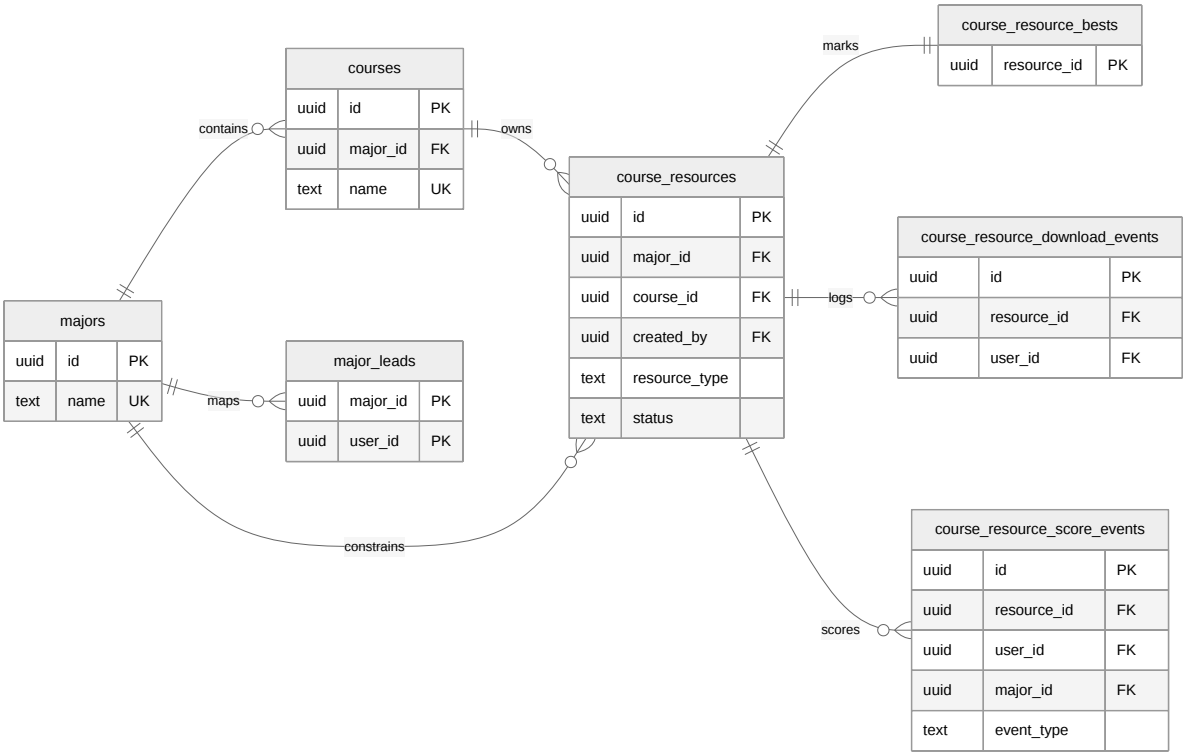


图 5　课程资源分享子系统 E-R 图

3.5　E-R 模型到关系模式的转换

概念结构转换为关系模式时，本设计按几条规则处理：一对一联系优先使用共享主键，例如 profiles.id 对应 auth.users.id；一对多联系在“多”端设置显式外键，例如 facility_rooms.building_id 和 courses.major_id；多对多联系拆成桥接表，例如 user_roles、role_permissions 和 major_leads；层级结构同时保留自引用和闭包表；事实表使用独立主键或复合业务唯一约束，不与主体表混用字段。这样转换后，后续说明候选键、函数依赖、索引、触发器和事务边界时会更清楚。

3.6　主要关系模式设计

为了便于后续详细设计，本节选取三条主线中最主要的关系模式进行汇总，如表 5 至表 7 所示。

表 5　平台基础子系统主要关系模式

关系模式	主键/候选键	设计要点
profiles	主键 id；候选键 student_id	与 auth.users 共享主键，保存业务资料和认证快照。
departments	主键 id	通过自引用外键构成部门树。
department_ closure	复合主键 (ancestor_id, descendant_id)	派生关系表，用于高效展开部门子树。
user_ departments	复合主键 (user_id, department_id)	将用户与部门的多对多关系拆分为独立桥接表。
user_roles	复合主键 (user_id, role_id)	保存用户与角色之间的授权关系。
role_ permissions	复合主键 (role_id, permission_id)	保存角色与权限之间的授权关系。
role_data_ scopes	复合主键 (role_id, module)	表达模块维度上的数据范围配置。
role_data_ scope_ departments	复合主键 (role_id,module, department_id)	保存自定义数据范围所对应的部门集合。
audit_logs	主键 id	采用 append-only 方式保存审计事实。

3.7　规范化分析

在本课程设计中，最容易破坏第一范式的多值事实，如用户角色、预约参与人和专业负责人等，都已经拆分为独立关系，没有用数组、逗号串或重复列存放在主体表中。所以，本节主要讨论第二范式、第三范式，以及它们与工程约束之间的平衡。

3.7.1　核心关系的函数依赖分析

表 8 先对几张代表性关系给出归纳结论，随后再说明判定过程。

先看 roles。该关系中 id 是主键，code 由唯一索引保证唯一，因此二者都可以作为候选键。于是存在 id → code,name,description,created_at,updated_at 和 code → id,name,description,created_at,updated_at。除这两类依赖之外，并不存在由非主属性决定其他属性的情形，因而该关系中所有非平凡函数依赖的决定因素都是候选键，可以按 BCNF 判断。

再看 user_roles 和 facility_reservation_participants 这两张桥接表。它们的共同点是：

表 6　功能房预约子系统主要关系模式

关系模式	主键/候选键	设计要点
facility_buildings	主键 id；业务唯一 name	保存楼房主数据。
facility_rooms	主键 id；业务唯一 (building_id,floor_no,name)	保存房间主数据，并依附楼房存在。
facility_reservations	主键 id	保存预约时间区间、状态流转和审核字段。
facility_reservation_participants	复合主键 (reservation_id, user_id)	保存参与人集合及申请人标记。
facility_bans	主键 id	独立保存封禁历史，不与用户主体混表。

表 7　课程资源分享子系统主要关系模式

关系模式	主键/候选键	设计要点
majors	主键 id；业务唯一 name	保存专业主数据。
major_leads	复合主键 (major_id, user_id)	保存专业负责人对应关系。
courses	主键 id；业务唯一 (major_id,name)	保存课程主数据。
course_resources	主键 id	资源主体表，保留受控冗余字段 major_id。
course_resource_bests	主键 resource_id	保存已发布资源的最佳推荐标签。
course_resource_download_events	主键 id	记录资源下载事实。
course_resource_score_events	主键 id；业务唯一 (user_id,resource_id, event_type)	记录积分事件事实。

表 8　代表性关系的函数依赖与范式分析

关系	候选键	关键函数依赖	分析结论
roles	id, code	id → code,name, description,created_at, updated_at；code → id,name,description, created_at,updated_at	每个决定因素都是候选键，达到 BCNF
user_roles	(user_id,role_id)	(user_id,role_id) → created_at	非主属性完全依赖整个键，不存在部分依赖和传递依赖，达到 BCNF
role_data_ scopes	(role_id,module)	(role_id,module) → scope_type,created_at, updated_at	在当前属性集合下可视为 BCNF
facility_ reservation_ participants	(reservation_id, user_id)	(reservation_id,user_id) → is_applicant,created_at	桥接关系与附加业务约束分离，达到 BCNF
courses	id；活动数据上的业务键 (major_id,name)	id → major_id,name, code,enabled,sort,\protect\ TU\textellipsis	以主键为核心不存在非主属性的传递依赖，达到 3NF

联系本身就是关系的主要语义，候选键分别为 (user_id,role_id) 和 (reservation_id,user_id)。在 user_roles 中，created_at 只能依赖完整的用户角色组合，而不会只依赖 user_id 或 role_id 的任一部分，因此不存在部分依赖。在 facility_reservation_participants 中，is_applicant 与 created_at 同样由完整键决定；数据库额外设置的部分唯一索引只是表达“同一预约至多一名申请人”这一业务规则，并没有改变该关系的主要函数依赖结构。因此，这两张表都属于拆分充分、范式较高的联系关系。

继续看 role_data_scopes。该关系以 (role_id,module) 为主键，其非主属性只有 scope_ type、created_at 和 updated_at。由于一个角色可以对应多个模块，一个模块也可以被多个角色配置，因此 role_id 与 module 任一单独属性都不足以决定 scope_type。真正的决定因素只能是二者的组合。这说明 scope_type 对键不存在部分依赖，也不存在经由其他非主属性形成的传递依赖。从当前属性集合来看，这一关系同样可以按 BCNF 理解。

最后看资源主数据中的 courses。该表在物理实现上以 id 为主键，同时用部分唯一索引保证“未逻辑删除课程”范围内 (major_id,name) 的业务唯一性。若从严格关系模式的角度讨论，可将 id 视为形式上的候选键，此时有 id → major_id,name,code,enabled,sort, remark,created_at,updated_at,deleted_at。表中并不存在 name 决定 major_id，或者 major_id 再决定其他非主属性的链式依赖，因此不会产生典型的传递依赖异常。也就是说，courses 虽然比纯桥接表拥有更多业务属性，但其主数据组织方式仍然保持了第三范式所要求的依赖边界。

这些关系说明，平台基础和业务模块中的主数据表、字典表、桥接表，大多能在实现中给出明确的候选键和函数依赖，并达到 3NF；不少联系关系还可以按 BCNF 理解。当前设计不只是“表能用”，也对关系边界、键约束和依赖控制做了整理。

3.7.2 受控冗余与规范化取舍

规范化分析还需要区分“关系内部的依赖问题”和“跨关系的派生冗余问题”。先看 department_closure，它从单表内部看并不是低范式关系，因为候选键是 (ancestor_id, descendant_id)，并且只有 (ancestor_id, descendant_id) → depth 这一类主要依赖。就表内结构而言，它同样可以保持较高范式。需要额外解释的是，它在全库语义上把 departments 中原本可经递归查询得到的祖先–后代可达关系提前物化了。也就是说，它是为层级范围查询服务的派生关系表，是有意识的空间换时间，而不是普通业务表中的随意重复。

其次，audit_logs 中的 actor_name、actor_email 和 actor_roles 也不能简单归为范式错误。若仅从 audit_logs 自身出发，以 id 为主键时，大部分属性都可以视为对主键的直接依赖；特殊之处在于，这几项字段与用户资料表、角色关系表之间形成了跨关系的快照冗余。保留这些字段不是为了省去联表，而是为了固定“操作发生当时”的历史语义，使审计记录不受用户后续改名、改邮箱或角色调整的影响。这一设计更看重历史真实性和可追溯性。

资源模块中的受控冗余也需要单独说明。按照严格的函数依赖链，courses.id 可以决定 courses.major_id。所以，在 course_resources 中同时保存 course_id 与 major_id，会形成“id → course_id → major_id”这一可以由联表推导出的依赖链。类似地，在 course_resource_score_events 中，资源主体又会决定 major_id 和 user_id。如果单纯以消除冗余为目标，这些字段完全可以不落表，而在查询时经由课程或资源主体推导得到。本课程设计保留这些派生属性，是因为按专业筛选、排行榜统计，以及“资源专业归属”“资源作者归属”等规则需要由数据库直接裁决。为避免由此带来更新异常，数据库又用 course_resources_course_major_fk 以及 course_resource_score_events_resource_major_fk、course_resource_score_events_resource_user_fk 将派生属性与源关系强绑定。这样处理后，冗余没有无控制地扩散，而是变成了可以被数据库对象验证的一致性设计。

本课程设计在规范化上的处理原则是：主数据和联系关系尽量保持 3NF 或 BCNF；确实需要保留的冗余，必须说明来源、用途和数据库层面的控制机制。这样既能对应教材中的范式分析，也能兼顾 PostgreSQL 实际使用中的性能、查询便利性和一致性。

3.8 物理对象规划概览

除表结构本身外，本轮数据库设计还包含多种数据库对象，如表 9 所示。

表 9 数据库对象规划概览

对象类型	代表对象	设计作用
索引	部门闭包索引、资源状态索引、预约时间窗口索引、审计时间线索引	支撑高频查询和冲突检测
唯一约束/部分唯一索引	角色编码唯一、资源去重唯一、未撤销封禁唯一、申请人唯一	保证业务唯一性并兼容软删除或状态过滤
检查约束	学号格式、预约时间合法性、资源字段互斥、状态一致性	直接拒绝非法数据
触发器/函数	部门防环、闭包维护、审计阻断、资料同步、最佳推荐状态控制	保证跨表或跨行规则
排斥约束	facility_reservations_room_active_time_excl	解决同房间活跃预约时间重叠
复合外键	数据范围父子配置、资源专业一致性、积分作者一致性	保持受控冗余与层级配置的一致性

表 9 汇总了本次物理设计使用的主要对象。索引、唯一约束、检查约束、复合外键、触发器和排斥约束分别处理完整性控制、查询组织和事务处理中的不同问题。

第四章　详细设计

4.1　平台基础子系统详细设计

4.1.1　新用户创建与资料同步流程

平台基础层先处理用户创建与资料同步。当前系统以 auth.users 作为身份事实源。数据库通过触发器函数 handle_new_user()、handle_user_email_confirmed() 和 sync_profile_from_auth_user() 自动维护 profiles。流程见图 6。

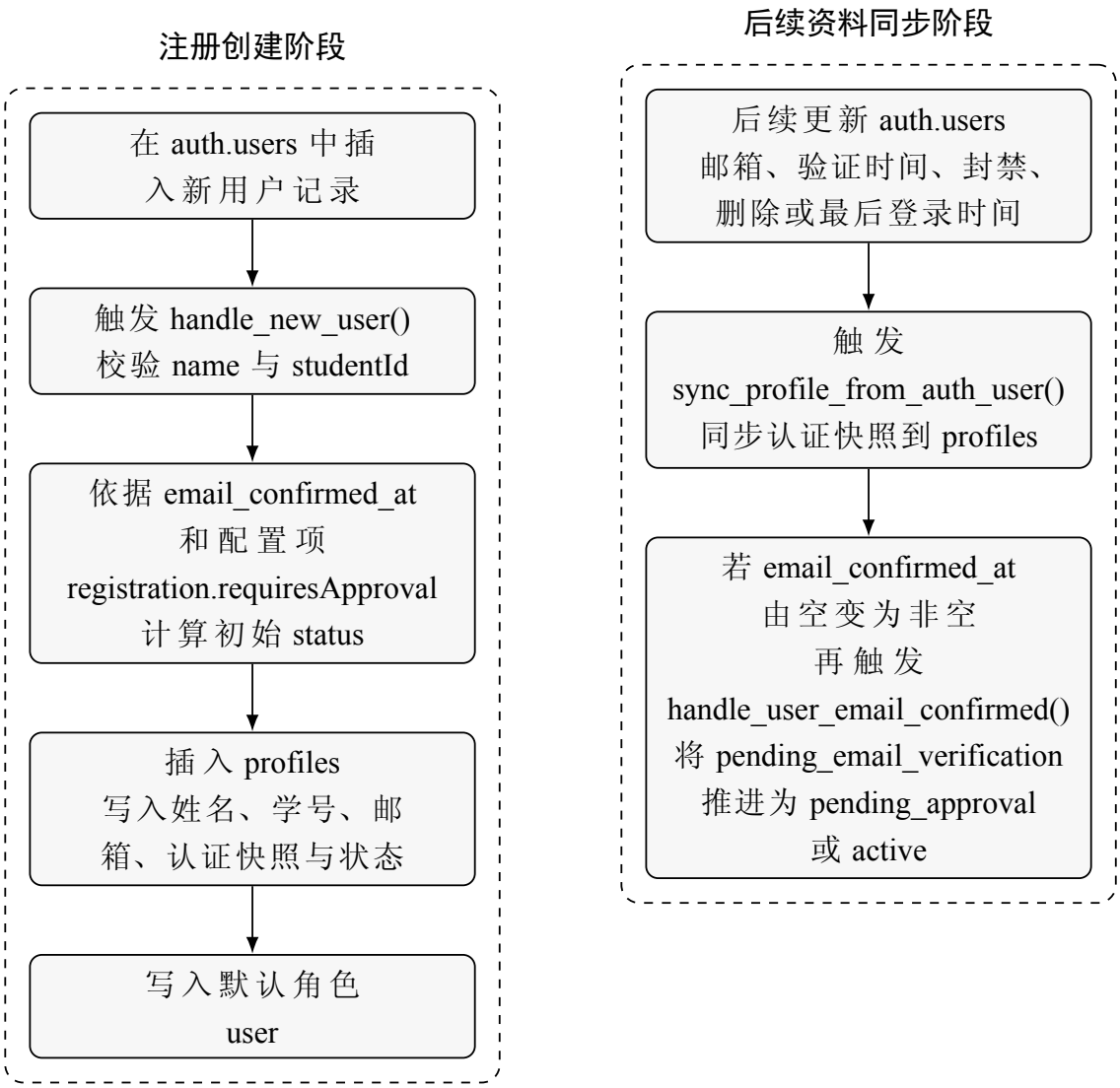


图 6　新用户创建与资料同步流程

这个流程说明，用户业务状态由数据库依据认证事实与配置开关推导；同时，profiles 中新增的邮箱、邮箱确认时间、封禁快照和删除快照字段，也会参与资料同步，用来保持业务资料与认证事实的一致性。

4.1.2 部门树维护与闭包表更新

部门树维护既涉及数据正确性，也涉及查询效率。系统通过 departments_assert_no_cycle()、departments_closure_after_insert()、departments_closure_after_update_parent() 三个函数完成防环和闭包表增量维护。图 7 给出了该过程。

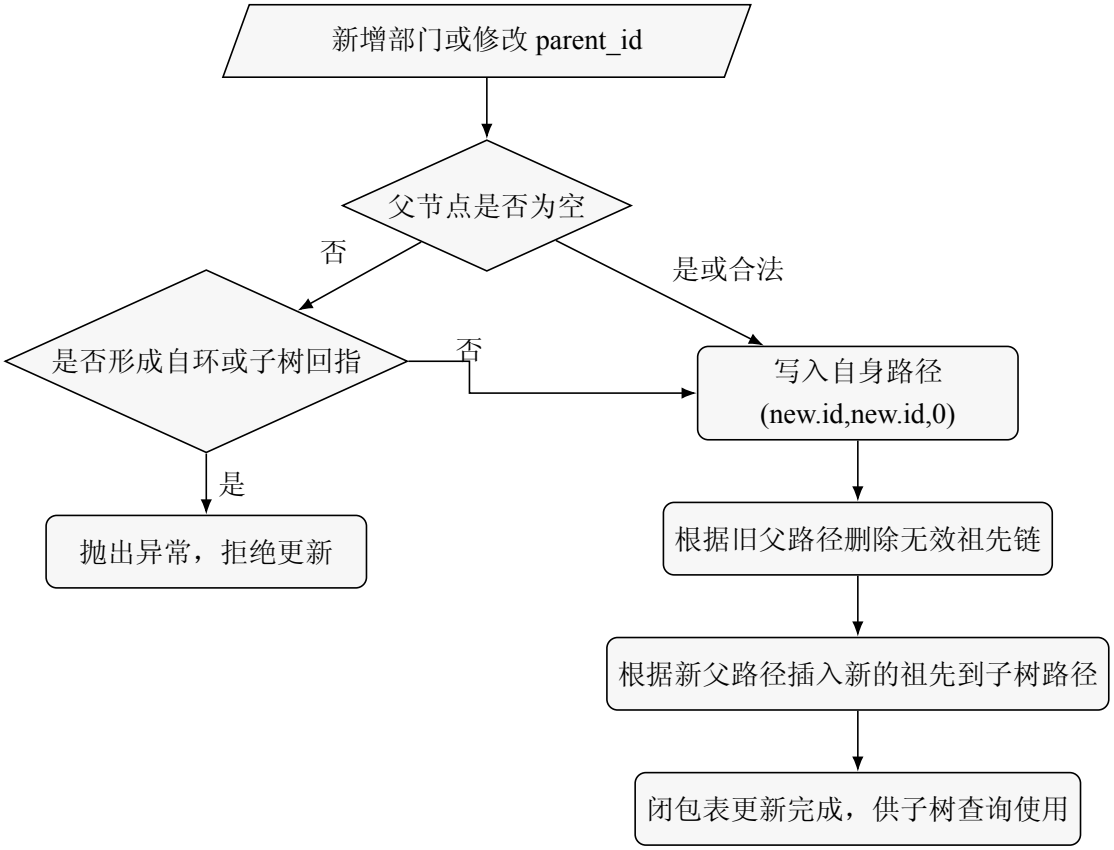


图 7　部门树闭包表维护流程

算法 1: 部门闭包表更新算法

输入: 待插入或待修改父节点的部门记录 new

输出: 闭包表 department_closure 维持一致

if new.parent_id 为空 **then**
    向闭包表插入 (new.id, new.id, 0)
    return
if new.parent_id = new.id 或 new.parent_id 位于 new.id 的后代集合中 **then**
    抛出“部门层级成环”异常并终止事务
若为新增部门，则把新父节点的全部祖先路径复制到新节点
若为移动部门，则删除旧祖先到该子树的无效路径，再按新父节点重建祖先路径
提交后，后续“部门及子部门”查询可直接命中闭包表与索引

与递归计算子树的方式相比，该算法虽然额外引入了派生关系表和维护触发器，但能够降低子树展开与组织树查询的代价。

4.1.3 角色数据范围整体替换事务

在权限配置场景中，系统采用“删旧 + 插新”的事务方式整体替换某角色的模块数据范围，避免父表与子表之间出现中间不一致状态。代表性 SQL 见代码 1。

Listing 1　角色数据范围整体替换事务

```
begin;

delete from public.role_data_scope_departments
where role_id = :role_id;

delete from public.role_data_scopes
where role_id = :role_id;

insert into public.role_data_scopes(role_id, module, scope_type)
values
  (:role_id, 'user', 'dept_and_child'),
  (:role_id, 'notice', 'custom');

insert into public.role_data_scope_departments(role_id, module, department_id)
values
  (:role_id, 'notice', :dept_a),
  (:role_id, 'notice', :dept_b);

commit;
```

代码 1 能安全执行，主要依赖两类约束：role_data_scopes.module 必须先通过 data_scope_modules 能力表校验；role_data_scope_departments 通过复合外键依赖父表记录存在。这些约束使事务在保持原子性的同时具有明确的参照边界。

4.2 功能房预约子系统详细设计

4.2.1 预约创建与并发控制流程

功能房预约模块最需要处理的是两件事：并发条件下同一房间的有效预约不能发生时间重叠，参与人集合也必须合法。图 8 给出了预约创建流程。

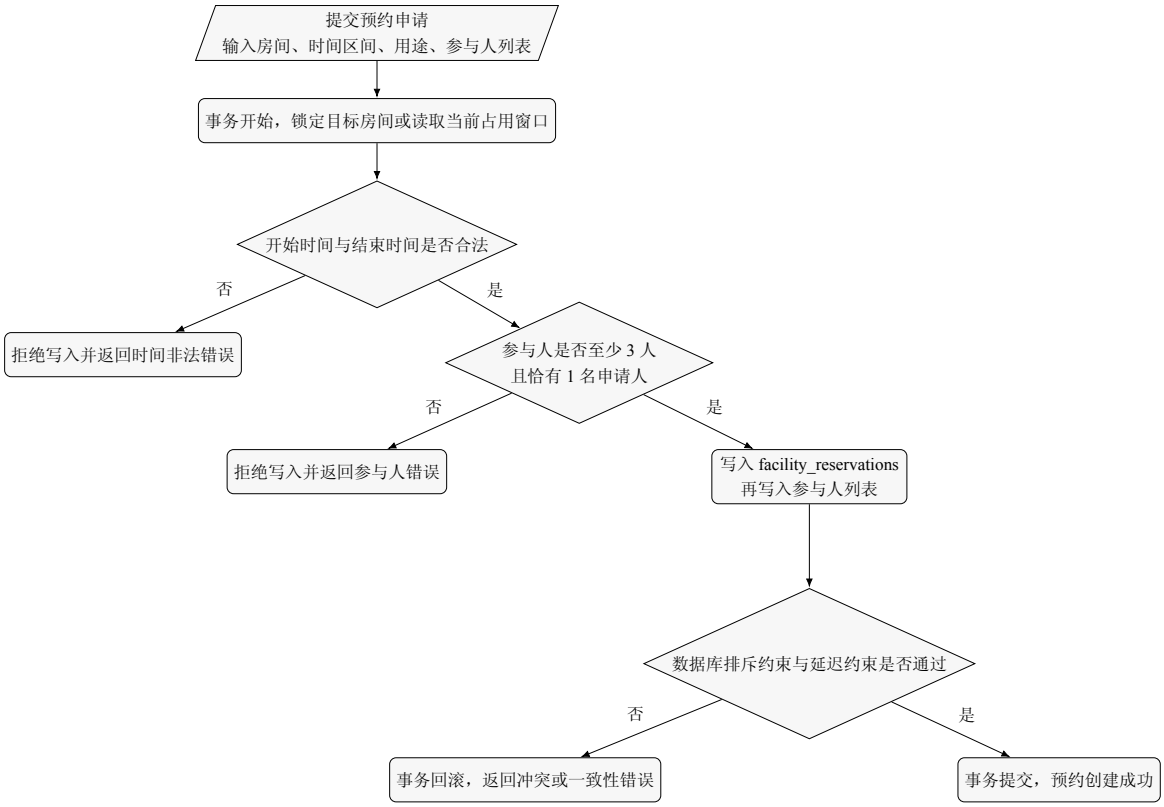


图 8　功能房预约创建与并发控制流程

算法 2: 功能房预约创建算法

输入: 房间编号、申请人、起止时间、用途、参与人集合

输出: 合法预约被提交；非法预约被数据库拒绝

```
开启事务
校验 end_at > start_at
写入 facility_reservations 主体记录，状态置为 pending
批量写入 facility_reservation_participants
在事务提交阶段，由延迟约束触发器检查参与人数、申请人数量和申请人与主表一致性
由 facility_reservations_room_active_time_excl 检查同房间活跃时间区间是否重叠
if 任一约束失败 then
    回滚事务并返回数据库错误信息
提交事务
```

这种“应用层事务 + 数据库约束”的写法，把并发场景下的最终一致性交给数据库检查。即便两个用户几乎同时为同一房间提交重叠预约，排斥约束仍可保证最终仅有一笔写入成功。

4.2.2 预约核心 DDL 与 DML 设计

功能房预约模块的代表性 DDL 片段如代码 2 所示。

Listing 2 预约模块代表性 DDL

```
alter table public.facility_reservations
  add constraint facility_reservations_room_active_time_excl
  exclude using gist (
    room_id with =,
    tstzrange(start_at, end_at, '[)') with &&
  )
  where (status in ('pending', 'approved'));

create unique index if not exists facility_reservation_participants_applicant_uq
  on public.facility_reservation_participants(reservation_id)
  where is_applicant;
```

这段 DDL 把“时间冲突”和“单一申请人”两项规则直接写进数据库对象。相应地，DML 操作也围绕这些约束展开，例如：

1. 创建预约时先写主体，再写参与人；
2. 审核通过时要求 reviewed_by 与 reviewed_at 同时落库；
3. 驳回时要求额外写入 reject_reason；
4. 取消时要求写入 cancelled_by 与 cancelled_at。

facility_reservations_status_consistency_chk 则负责把不同状态下辅助字段的组合合法性明确表述为数据库规则。

4.2.3 典型复杂查询

预约模块的典型查询之一，是按时间窗口判断房间占用情况。相应 SQL 见代码 3。

Listing 3 查询某房间在时间窗口内的占用情况

```
select r.id,
       r.room_id,
       r.status,
       r.start_at,
       r.end_at
from public.facility_reservations r
where r.room_id = :room_id
  and r.status in ('pending', 'approved')
  and r.start_at < :window_end
  and r.end_at > :window_start
order by r.start_at asc, r.id asc;
```

该查询与 facility_reservations_room_active_time_idx 部分索引相对应，说明索引设计是依据高频时间窗口查询配置的。

4.3 课程资源分享子系统详细设计

4.3.1 资源投稿、审核与发布流程

课程资源模块包含较完整的状态流转过程，核心流程见图 9。

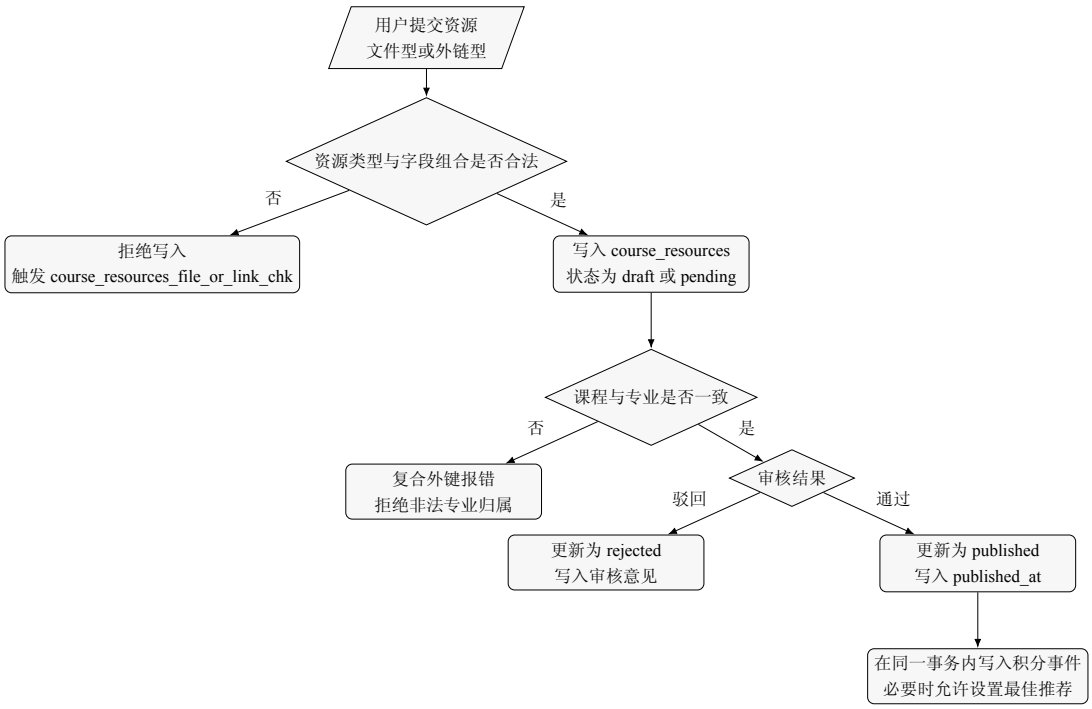


图 9　课程资源投稿、审核与发布流程

图 9 可以看出，资源投稿并不是单一的“插入一条记录”操作，而是字段合法性、课程归属一致性、审核状态更新与积分事件写入共同组成的连续过程。这里最容易出错的两处边界，是资源类型与字段组合检查，以及课程与专业归属的一致性校验；前者约束资源主体自身的字段结构，后者约束资源主体与教学主数据之间的对应关系。只有这两层校验都成立，后续的发布、推荐与积分统计才有可信的数据基础。

算法 3: 课程资源审核与积分写入算法

输入: 资源编号、审核人、审核结果、审核意见

输出: 资源状态与积分事件保持一致

```
开启事务
读取资源主体及其当前状态
if 资源与课程、专业归属不一致 then
    由复合外键拒绝提交
if 审核通过 then
    更新 status='published'、reviewed_at、published_at
    插入 course_resource_score_events(event_type='approve')
else
    更新 status='rejected'、reviewed_at、review_comment
if 后续设置最佳推荐 then
    仅当资源状态为 published 时允许写入 course_resource_bests
提交事务
```

4.3.2 资源核心约束与受控冗余

课程资源模块的代表性数据库约束如代码 4 所示。

Listing 4　资源模块代表性 DDL

```
alter table public.course_resources
  add constraint course_resources_course_major_fk
  foreign key (course_id, major_id)
  references public.courses(id, major_id)
  on update cascade
  on delete restrict;

alter table public.course_resource_score_events
  add constraint course_resource_score_events_resource_user_fk
  foreign key (resource_id, user_id)
  references public.course_resources(id, created_by)
  on update cascade
  on delete cascade;
```

代码 4 中，受控冗余由复合外键维持一致性。借助该约束，course_resources.major_id 被限制为课程所属专业；course_resource_score_events.user_id 被限制为资源作者。这样可以在保留统计和授权过滤所需字段的同时，控制冗余带来的一致性风险。

4.3.3 最佳推荐与状态触发器

最佳推荐依附于资源发布状态而存在，可视为资源发布态上的派生事实。数据库使用两个触发器函数完成该项约束：

1. course_resource_best_requires_published()：在插入或更新最佳记录前检查资源是否已发布；
2. course_resource_drop_best_when_not_published()：当资源状态变为非 published 时自动删除最佳记录。

这两个触发器把“最佳推荐仅作用于已发布资源”这一规则落在数据库层。

4.3.4 典型聚合查询

课程资源模块的典型聚合查询之一是积分排行榜，见代码 5。

Listing 5 课程资源积分榜查询

```
select s.user_id,
       p.name,
       sum(s.delta) as score,
       sum(case when s.event_type = 'approve' then 1 else 0 end) as approve_count,
       sum(case when s.event_type = 'best' then 1 else 0 end) as best_count
from public.course_resource_score_events s
join public.profiles p
  on p.id = s.user_id
where (:major_id is null or s.major_id = :major_id)
group by s.user_id, p.name
order by sum(s.delta) desc, p.name asc
limit 20;
```

该查询直接对积分事实表进行聚合，便于表达“首次通过”“首次最佳”等业务语义，也为后续审计与排行榜扩展保留了空间。

4.4 SQL 设计汇总

结合前三节，本次课程设计中有代表性的 SQL 设计可以分为三类。第一类是 DDL，主要处理显式物理外键、复合外键、检查约束、部分唯一索引、排斥约束和触发器等数据库对象；第二类是 DML，主要处理预约创建、状态审核、最佳推荐设置、角色数据范围整体替换等事务性写操作；第三类是复杂查询，主要处理部门子树展开、房间时间窗口占用查询、审计时间线查询以及下载榜、积分榜的聚合统计。

这三类 SQL 不是互不相关的片段，而是分别承担结构定义、状态写入和结果读取三种职责。关系模式、约束、索引、触发器和这些 SQL 配合起来，数据库详细设计才从表结构落到实际运行结果。

第五章　设计结果/运维

本章用系统运行截图说明平台基础、功能房预约和课程资源分享三个子系统的典型操作结果。各图均取自测试环境中的真实页面，重点看数据库设计如何落到页面状态、筛选条件、记录列表和统计结果上。

5.1　平台基础子系统运行结果

5.1.1　新用户创建与资料同步结果

图 10 是后台“用户”管理页面的实际运行结果。

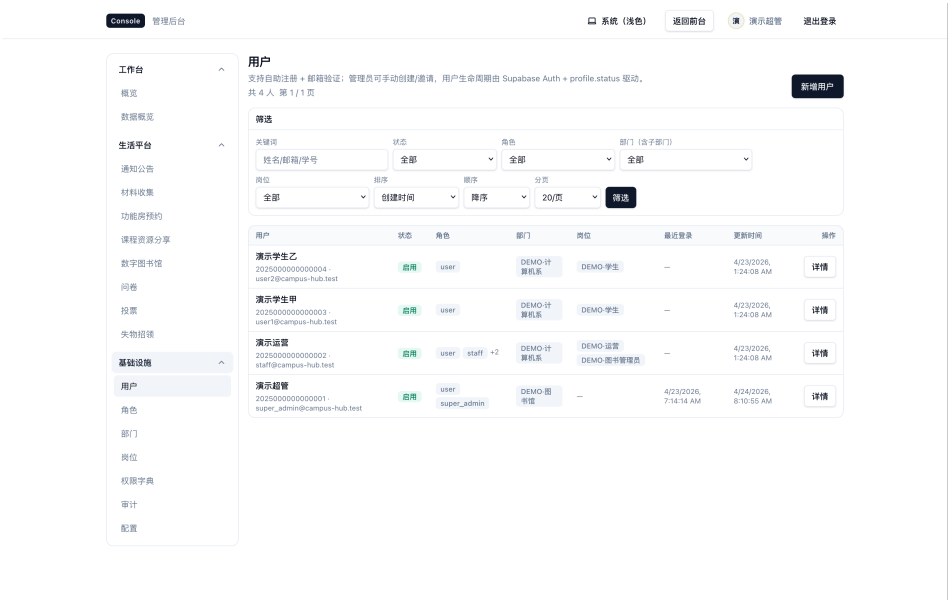


图 10　新用户创建后资料同步结果

图中页面上方是按姓名、邮箱、学号、状态、角色、部门和岗位筛选的条件区，下方列表显示用户姓名、学号或邮箱、当前状态、角色、所属部门、岗位、最近登录时间和更新时间等信息。四条演示数据能够稳定出现在同一列表中，说明新用户创建后，身份、资料和组织归属信息已经完成同步，并可被后台统一检索与管理；数据库中的用户主记录、扩展资料和角色组织关联，也能共同支撑管理页面展示。

5.1.2　部门树维护结果

图 11 给出了部门管理页面。

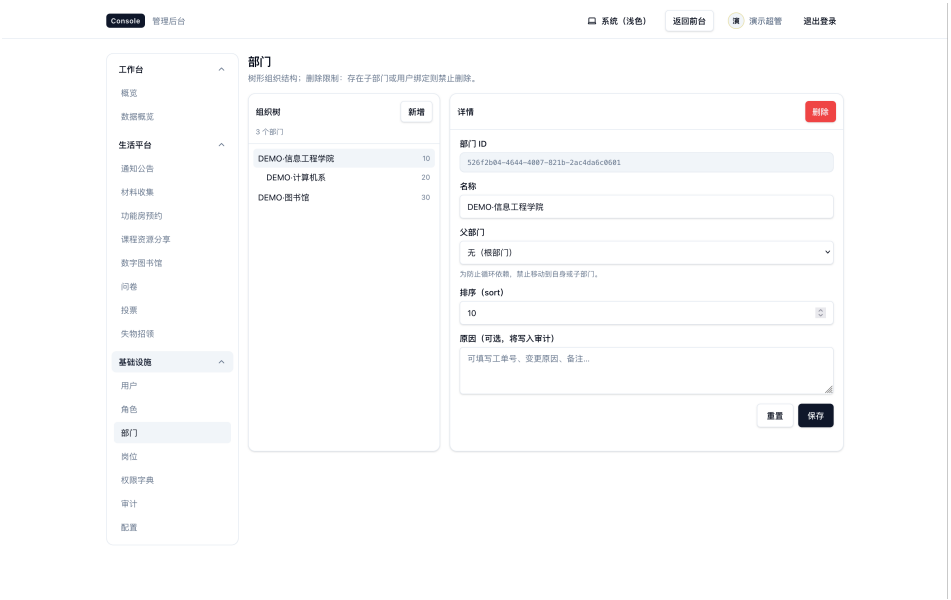


图 11　部门树维护结果

左侧“组织树”区域列出了三个部门节点，当前选中的节点为“DEMO: 信息工程学院”；右侧详情区给出了部门 ID、名称、父部门、排序以及变更原因等维护字段，并提供保存与删除操作。页面顶部明确提示存在子部门或用户绑定时禁止删除，这与数据库中的层级关系约束和引用约束保持一致。部门树结构已经能够在页面中稳定呈现，部门层级、父子关系和维护字段也具备可操作的落地形式，为后续按部门做范围控制与统计查询提供结构基础。

5.1.3　角色配置结果

图 12 是“工作人员”角色的编辑窗口。

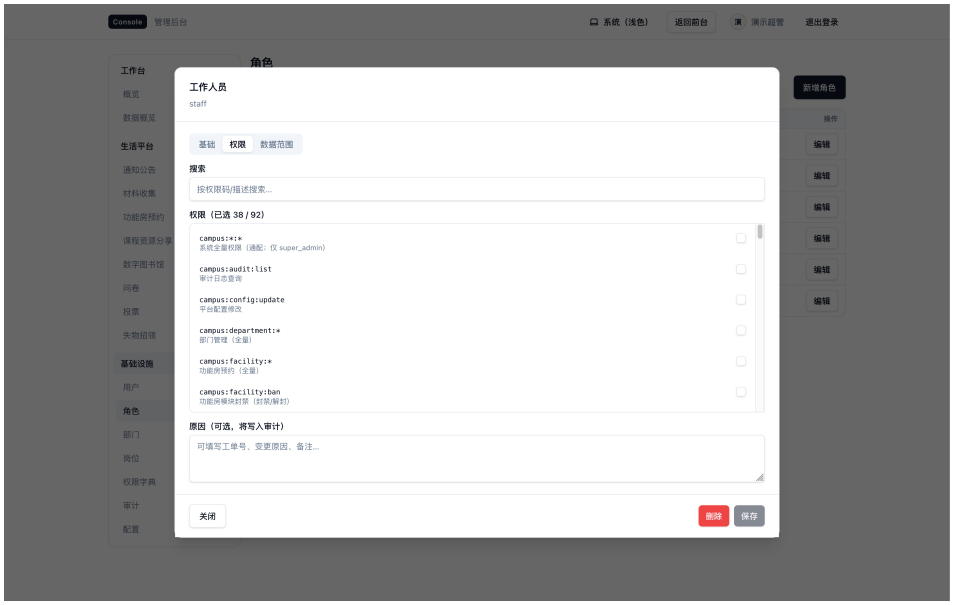


图 12　角色配置结果

该窗口包含基础、权限和数据范围三个配置入口。当前截图位于权限页，页面显示已选权限数量为 38 / 92，并列出 campus:audit:list、campus:department:*、campus:facility:* 等权限编码及中文说明。窗口底部还有审计原因输入框和保存按钮，说明角色配置不只是前端勾选，还会写入权限关系并留下审计记录。数据库中保存的角色授权信息可以被管理端回显，并支持后续编辑和追踪。

5.2 功能房预约子系统运行结果

5.2.1 合法预约提交结果

图 13 是功能房预约前台的纵览页面。

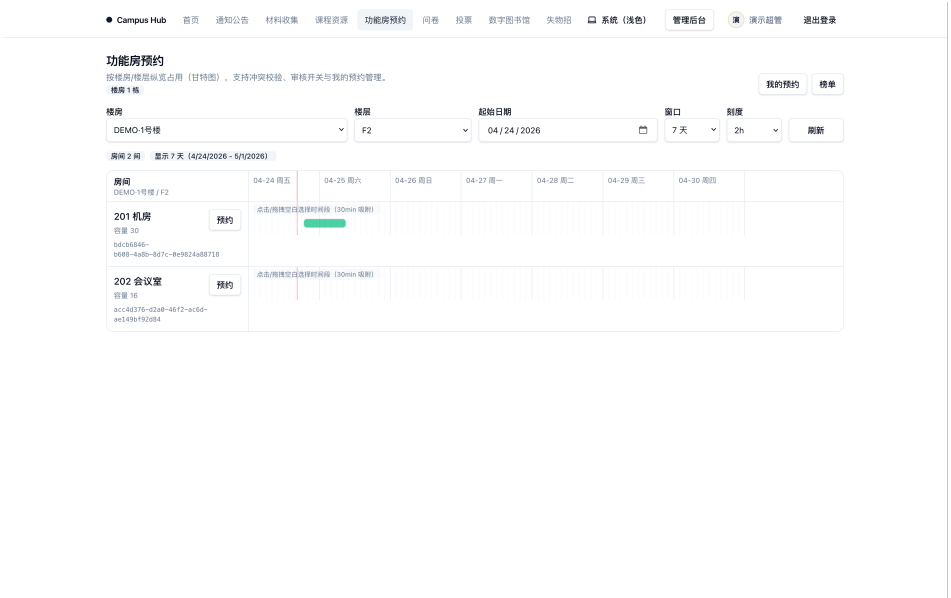


图 13　合法预约提交结果

页面当前选中“DEMO:1 号楼”与 F2 楼层，起始日期为 2026-04-24，展示窗口为 7 天、时间刻度为 2 小时。房间列表中给出了 201 机房和 202 会议室两间房间的名称、容量与编号信息，其中 201 机房的时间轴上已经出现一段绿色预约条，表示该房间存在一笔已经成功写入并参与占用展示的预约记录。合法预约提交后，房间、时间区间和占用状态能够被前台甘特图式页面正确读取，预约记录也已经进入可查询的业务状态。

5.2.2 冲突预约拦截结果

图 14 是在相同房间、相同时间窗口内再次尝试预约时的拦截结果。

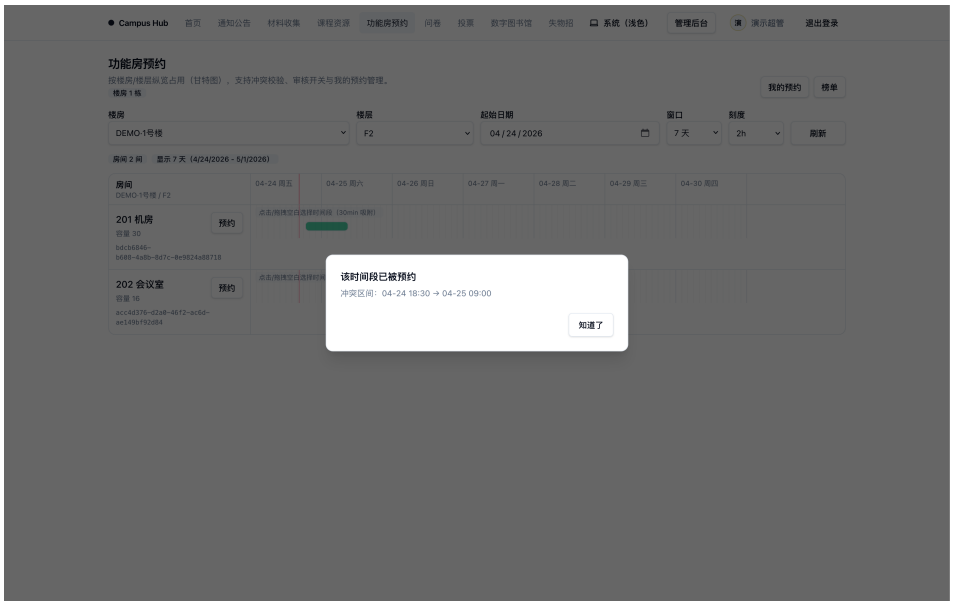


图 14　重叠预约被拒绝的运行结果

页面弹窗直接给出“该时间段已被预约”的提示，并明确标识冲突区间为 04-24 18:30 至 04-25 09:00。相比只返回失败状态，这个弹窗把数据库中的时间冲突判断转成了用户能看懂的提示。预约冲突不是只靠业务层约定，而是已经由数据库约束拦住非法写入。

5.2.3　预约审核后的状态变更结果

图 15 给出了预约审核页面。

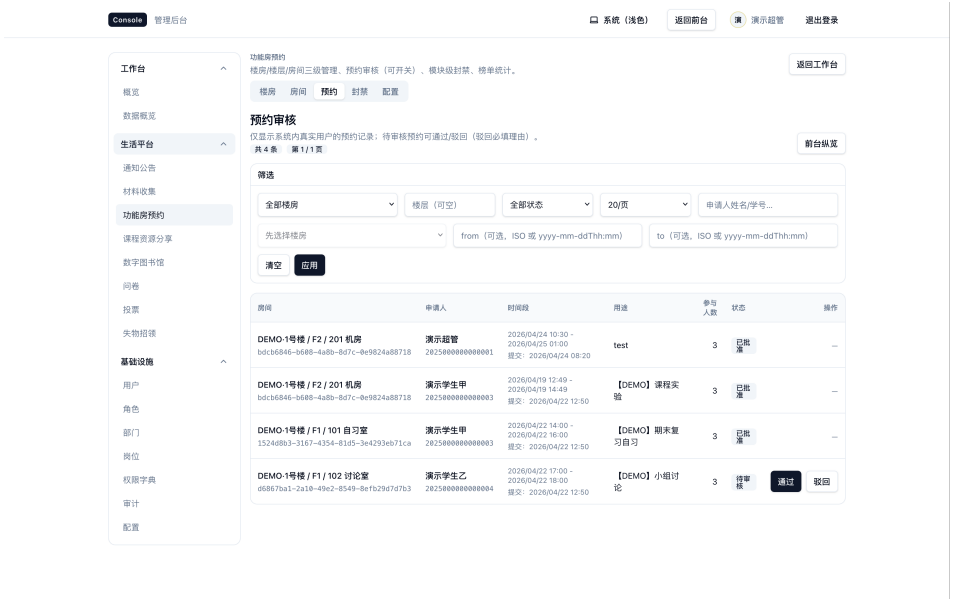


图 15　预约审核后的状态变更结果

页面上方提供了楼房、楼层、状态、分页、申请人以及时间区间等筛选条件，列表中逐条列出房间、申请人、预约时段、用途、参与人数和当前状态。图中前三条记录已

经显示为“已批准”，末条记录仍处于“待审核”状态，并同时给出了“通过”和“驳回”操作按钮，页面说明中还明确写出了驳回需要填写理由。预约记录在提交后能够进入审核队列，审核完成后再以不同状态回到列表页面，数据库中的状态流转已经被管理端页面直接呈现出来。

5.3 课程资源分享子系统运行结果

5.3.1 资源投稿与审核发布结果

图 16 是一条课程资源在审核通过并发布后的详情页。



图 16　资源投稿与审核发布结果

页面顶部给出资源标题“[DEMO] MDN 参考手册”、课程与专业归属“DEMO: 软件工程 / 数据库系统”，并用“已发布”“外链”标签标识状态和资源类型。页面中部列出原始 URL、规范化 URL、提交时间、审核时间、审核意见、发布时间、作者、下载次数和更新时间等信息。资源投稿、审核和发布后的主要字段可以在同一详情页回显，资源状态、作者信息和访问统计也能从数据库中正确读出。

5.3.2 课程资源积分榜查询结果

图 17 是课程资源榜单页面。

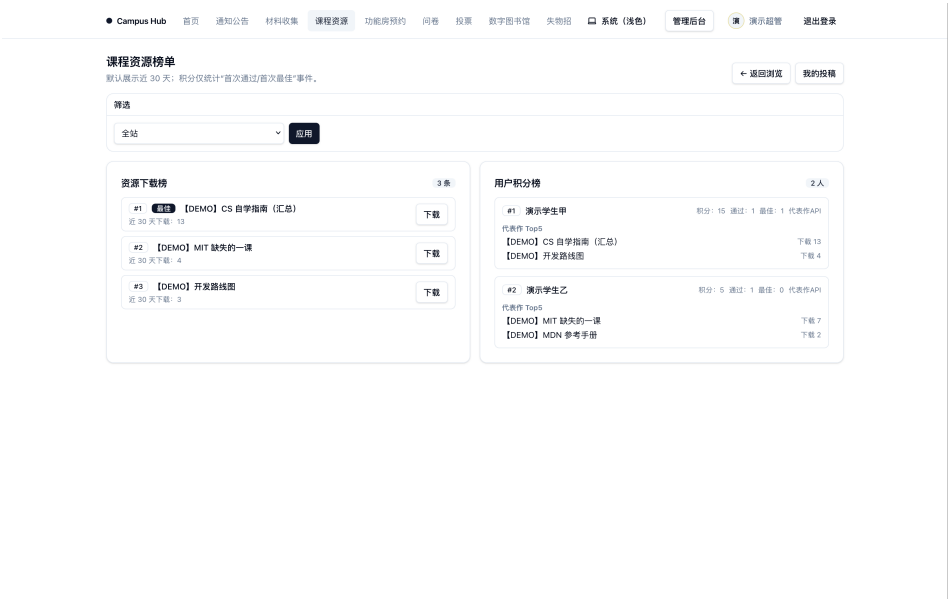


图 17　课程资源积分榜查询结果

页面标题说明榜单默认统计近 30 天，积分只计入“首次通过”和“首次最佳”事件。左侧“资源下载榜”列出资源名称、是否为最佳资源和近 30 天下载次数；右侧“用户积分榜”给出用户积分、通过次数、最佳次数和代表作品 Top5。图中演示学生甲以 15 分位居第一，演示学生乙以 5 分位居第二。下载事件和积分事件可以被聚合成排行榜，并在业务页面中使用。

5.4　本章小结

本章列出了平台基础、功能房预约和课程资源分享三个子系统的运行结果。用户列表、部门树、角色配置、预约纵览、冲突提示、审核列表、资源详情和积分榜单，分别对应前文的关系模式、约束设计和统计查询。这些页面说明，数据库设计已经进入实际业务流程，而不是只停留在表结构说明中。

第六章　总结

6.1　主要收获

本次课程设计围绕平台基础、功能房预约和课程资源分享三个子系统，完成了需求分析、E-R 建模、关系模式转换、约束设计、SQL 实现和运行验证。完成这一轮梳理后可以看到，数据库设计不是系统开发后期补上的实现细节，而是直接决定业务对象怎么表达、业务规则怎么约束、运行结果怎么验证的部分。前文中的关系模式、主外键结构、索引方案、触发器机制和统计查询，也不是零散的技术点，而是三个子系统能够正常运行的数据库基础。

从建模与规范化的角度看，本次设计让我对几个基础问题有了更具体的认识。多对多关系需要用桥接表单独表达，这样参与对象和关联事实才不会混在一起；树形组织结构可以用自引用表达上下级关系，也可以用闭包表提高范围查询效率；第三范式并不等于所有重复字段都不能出现，关键是冗余字段是否有明确用途，数据库能否持续维护它的正确性。范式分析的目的不是机械删掉所有重复信息，而是在结构清楚、依赖明确和一致性可控之间取平衡。

从 PostgreSQL 的具体实现看，一些教材中的概念在本次设计中有了可运行的对应物。功能房预约的时间冲突控制使用 EXCLUDE USING gist，可以直接处理时间区间互斥；参与人集合中“至少 3 人且恰有 1 名申请人”的规则，适合用延迟约束触发器检查；课程资源模块中的复合外键和受控冗余也说明，统计需求不一定会破坏结构，只要约束设计清楚，数据库仍然可以保证一致性。第五章的运行截图也证明，这些设计不是只停留在 DDL 中，而是已经对应到页面上的查询、审核和拦截结果。

6.2　当前不足

本次设计仍有一些不足。高频统计查询目前主要依赖事实表、索引和聚合语句，可以满足当前页面展示；但数据量继续增大后，榜单、统计和后台分析查询可能会有性能压力，后续可以把一部分稳定统计逻辑整理成视图或物化视图。功能房封禁模块已经用 expires_at 和 revoked_at 表达封禁状态，但“已自然过期”“人工撤销”“仍在生效”这些判定还可以写得更清楚。审计日志目前采用快照式引用，可以保留操作发生时的上下文；如果以后要做跨主体、跨模块追踪，还需要补充专门的查询设计。

另外，报告已经说明了索引设计、复杂查询和约束协同关系，但对执行计划、成本估计和大规模数据下的性能变化还缺少实验支撑。也就是说，当前设计可以证明关系模式和业务规则在逻辑上成立，也能证明典型场景在现有数据规模下可以运行；如果要提高工程完整性，还需要补充性能测试、统计对象抽象和运维观测相关内容。这些不足不影响本次课程设计的主要结论，但也提醒我：数据库设计不是一次性完成的静态工作，后续业务扩展和数据增长都会继续反过来影响表结构、索引和约束。

6.3 后续改进方向

后续如果继续完善，我会先从三处入手。结构上，继续梳理封禁状态、审计追踪和统计对象的边界，把现在还依赖业务约定解释的语义尽量写进数据库结构。性能上，围绕排行榜查询、范围过滤查询和预约冲突检测准备更大规模的测试数据，补充执行计划、索引命中和查询耗时对比。运维上，增加视图、物化视图、定期统计维护策略，以及面向约束和触发器的巡检脚本。这样数据库后续不仅便于开发，也更便于验证、排障和维护。

参考文献

[1] 王珊, 杜小勇, 陈红. 数据库系统概论 [M]. 6 版. 北京: 高等教育出版社, 2023.

[2] 施伯乐, 丁宝康, 汪卫. 数据库系统教程 [M]. 3 版. 北京: 高等教育出版社, 2010.

[3] 孟小峰, 周龙骧, 王珊. 数据库技术发展趋势 [J]. 软件学报, 2004, 15(12): 1822-1836.

[4] 孟小峰, 马超红, 杨晨. 机器学习化数据库系统研究综述 [J]. 计算机研究与发展, 2019, 56(9): 1803-1820.

[5] 赖思超, 吴小莹, 彭煜玮, 彭智勇. 数据库索引调优技术综述 [J]. 计算机研究与发展, 2024, 61(4): 929-954.

[6] Codd E F. A relational model of data for large shared data banks[J]. Communications of the ACM, 1970, 13(6): 377-387.

[7] Chen P P-S. The entity-relationship model: toward a unified view of data[J]. ACM Transactions on Database Systems, 1976, 1(1): 9-36.

[8] Fagin R. Multivalued dependencies and a new normal form for relational databases[J]. ACM Transactions on Database Systems, 1977, 2(3): 262-278.

[9] Guttman A. R-trees: a dynamic index structure for spatial searching[C]//Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1984: 47-57.

[10] Hellerstein J M, Naughton J F, Pfeffer A. Generalized search trees for database systems[C]//Proceedings of the 21st International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann, 1995: 562-573.

[11] Gray J, Reuter A. Transaction processing: concepts and techniques[M]. San Mateo, CA: Morgan Kaufmann, 1992.

[12] Silberschatz A, Korth H F, Sudarshan S. Database system concepts[M]. 7th ed. New York: McGraw-Hill Education, 2019.

[13] Elmasri R, Navathe S B. Fundamentals of database systems[M]. 7th ed. Boston: Pearson, 2016.

[14] Hellerstein J M, Stonebraker M, Hamilton J. Architecture of a database system[J]. Foundations and Trends in Databases, 2007, 1(2): 141-259.

[15] PostgreSQL Global Development Group. Constraints[EB/OL]. PostgreSQL 16 Documentation. [2026-04-24]. https://www.postgresql.org/docs/16/ddl-constraints.html

[16] PostgreSQL Global Development Group. Multicolumn Indexes[EB/OL]. PostgreSQL 16 Documentation. [2026-04-24]. https://www.postgresql.org/docs/16/indexes-multicolumn.html

[17] PostgreSQL Global Development Group. Partial Indexes[EB/OL]. PostgreSQL 16 Documentation. [2026-04-24]. https://www.postgresql.org/docs/16/indexes-partial.html

[18] PostgreSQL Global Development Group. CREATE TRIGGER[EB/OL]. PostgreSQL 16 Documentation. [2026-04-24]. https://www.postgresql.org/docs/16/sql-createtrigger.html

[19] PostgreSQL Global Development Group. Range Types[EB/OL]. PostgreSQL 16 Documentation. [2026-04-24]. https://www.postgresql.org/docs/16/rangetypes.html

[20] PostgreSQL Global Development Group. btree_gist[EB/OL]. PostgreSQL 16 Documentation. [2026-04-24]. https://www.postgresql.org/docs/16/btree-gist.html